Structure and Use of AILabReal

Table of Contents

[Purpose 2](#_Toc2584556)

[Introduction 3](#_Toc2584557)

[Architecture 3](#_Toc2584558)

[Description 3](#_Toc2584559)

[Program Structure: 4](#_Toc2584560)

[Language and Libraries 4](#_Toc2584561)

[Python 4](#_Toc2584562)

[pygame 4](#_Toc2584563)

[numpy 4](#_Toc2584564)

[User Interface 4](#_Toc2584565)

[Map 4](#_Toc2584566)

[Main Menu 4](#_Toc2584567)

[Execution Controls 5](#_Toc2584568)

[Testing 5](#_Toc2584569)

[Test Commands 5](#_Toc2584570)

[Source Code 6](#_Toc2584571)

[Program Flow 6](#_Toc2584572)

[Files 6](#_Toc2584573)

[File DuneBuggy.py 6](#_Toc2584574)

[Program Start 6](#_Toc2584575)

[Description 6](#_Toc2584576)

[File Desertmap.py 6](#_Toc2584577)

[Description 6](#_Toc2584578)

[Classes 7](#_Toc2584579)

[File Button.py 7](#_Toc2584580)

[Description 7](#_Toc2584581)

[Class Button 7](#_Toc2584582)

[File Vehicle.py 8](#_Toc2584583)

[Description 8](#_Toc2584584)

[Method: Vehicle init 8](#_Toc2584585)

[Class Vehicle 8](#_Toc2584586)

[Class Hills 9](#_Toc2584587)

[Class Hill 9](#_Toc2584588)

[Def load\_png 9](#_Toc2584589)

[File DriverAI.py 9](#_Toc2584590)

[Description 9](#_Toc2584591)

[Class DriverAi 9](#_Toc2584592)

[File Sensors.py 10](#_Toc2584593)

[Description 10](#_Toc2584594)

[Classes 10](#_Toc2584595)

[File Tools.py 10](#_Toc2584596)

[Description 10](#_Toc2584597)

[Class Debug 10](#_Toc2584598)

[Class Write 10](#_Toc2584599)

[Class Help 11](#_Toc2584600)

[Class UnitTest 11](#_Toc2584601)

[Development Process 11](#_Toc2584602)

[Language 11](#_Toc2584603)

[Language is Python 3.7.0. 11](#_Toc2584604)

[Library pygame 11](#_Toc2584605)

[Install pygame. 11](#_Toc2584606)

[Library numpy 11](#_Toc2584607)

[Editing 11](#_Toc2584608)

[Execution 12](#_Toc2584609)

[Backup 12](#_Toc2584610)

# Purpose

The program AILabR (real math version) is Artificial Intelligence educational test bed for

1. Python graphics on Windows using pygame
2. Artificial Intelligence (AI)

This Python code demonstrates:

* AI – an artificial Intelligence engine that:
  + Uses neurons to make decisions
  + Applies adjustable complex weights to control actions
  + Uses feedback to update the current decision
  + Does not use learning – it is continuously adaptive
  + The AI structure Is easy to change
* Use of pygame for displaying a simulation
* Display of a map on a Windows screen
* Adds menu buttons to the map
* Moves an object across the screen

# Introduction

AILab is a simulation of an autonomous vehicle, i.e. a driverless car. The vehicle is presented on a screen that shows a map of a region. In the first instance the map is of a desert. The objective is for a vehicle to move from a starting point across the map to an exit. There is a destination, the exit, and obstacles such as hills. All motion is controlled by an Artificial Intelligence engine (AI) or keyboard arrows. The AI operates with complex vectors to represent angle and distance to objects on the map.

# Architecture

## Description

An AI or user-controlled object (looks like a dune buggy) is moved towards an exit.

Under AI control It avoids blocks (hills) detected by radar type sensors.

The observer can set/unset blocks.

The AI is continuously adaptive – it requires no learning. This is implemented by knowing the desired end value. Each AI neuron is fed data through variable weights. The current output is compared to the desired end value and the difference is fed back to the weights for the next round of computation.

Three sensors are mounted on the vehicle, one in front and one on each side at an angle of 45 degrees. A sensor is like a Lidar or Radar. It returns the vector sum of the location and power of every object it can see. In this case a sensor has a field of view that is a sine wave, like an antenna. It is down 3db (one half) at +/- 45 degrees and 0 at +/- 90 degrees.

A sensor output is (S + N) where S is the signal from the exit and N is the sum of all the noise signals, sum (Ni). A noise signal is from a block (hill). There is a sensor postprocessor that extracts the desired signal (S) so it can be differentiated from all the noise signals, Ni. The goal of the AI is to reduce S / (S + N) to 1 by removing the noise, N. The vehicle is constantly moved so that the return from N (the hills) is minimized and the return from S is maximized. Ideally, there is no N and the result is S/S or 1. Any difference from 1 is used to modify the weights. For those with a differential math background, this is taking the second partial WRT time and setting it to 0.

## Program Structure:

An icon of a dune buggy moves through a grid of cells mapped onto a valley. The objective is for the dune buggy to move from the entrance of the valley to the exit of the valley.

pygame – imported open source library to support screen, graphics and actors (see ….).

map – this release is implemented in a desert version per class desertmap.py. It includes a target exit and user placed blocks (hills).

vehicle - defines and moves the vehicle sprite.

dunebuggy - simulator that controls activity of a vehicle via AI or user input.

AI - an ANN (Artificial Neural Network) that guides the vehicle through obstacles

# Language and Libraries

The code is written in Python and uses the pygame (game display) and numpy (math) libraries to generate graphics and run the action.

This AI is derived from Milo Harper's source at <https://github.com/miloharper/simple-neural-network>. It is also based on an AI named AGIPA that I developed with others and demonstrated with hardware for the TDRSS satellite in 1975 – see <https://ieeexplore.ieee.org/document/1147439>. This paper was presented by my manager, James Smith.

## Python

Python can be installed from https://www.python.org/downloads.

## pygame

PyGame's official installation guide is <https://www.pygame.org/wiki/GettingStarted>.

# User Interface

## Map

A map of the valley is displayed as a grid of cells. The pattern of a cell denotes:

* Grassland - transparent cells represent flat grassland.
* Hills - gray cells represent impassible hills.
* E – represents the entrance
* X – represents the exit
* AILab - a 16x16 pixel icon (sprite) of a AILab

## Main Menu

The main menu is a list of the following buttons.

* Map – display the map of the valley, enable execution controls
* Exit – return to the operating system, prompts to save
* Save – store the current map, configuration, weights and tank position to the current file
* SaveAs – store the current map, configuration, weights and tank position to a new file
* Test – a file of test commands is executed. Select the file name from the displayed list.
* Log – toggle logging

## Execution Controls

* select : **Hills**: make the selected cell into a gray impassible hill. Hold the ‘h’ key down and drag the cursor to build a string of hill cells.
* select : **Grassland**: return the selected hill cell to flat grassland. Hold the ‘g’ key down and drag the cursor to build a string of grassland cells.
* ‘Esc : **Main** Menu: Press Esc to return to the main menu. Simulation is paused.
* Space : **Starts/Stops** automatic MI-based action. Start deactivates the arrow keys.
* Arrows : **Move** the AILab by one grassland cell (forward, reverse, left, right)
* ‘r’ : **Randomize** the AI weights

# Testing

dunebuggy.py sets up the screen. Before it begins to execute the main loop it checks the global unit\_test. If that is TRUE then the unittest() method in tools.py is called. It reads the dbtest.json file into a dictionary and executes tests per the commands in that file.

### Test Commands

grass x y – set the cell at x,y to grass

hill x y – set the cell at x,y to hill

db x y – set the dunebuggy at cell x,y

sensor – ID, angle

f – forward

b – back

l – left

r – right

step – AI moves forward one step

run – AI runs from current location till blocked or at exit

log on – record log statements to the log file

log off – deactivate logging to the log file

# Technical Background

Inputs:

Hills – Hi – block the path of the vehicle

Target – T – the complex signal from the map exit

Weight - Wi – complex weight modifies the value of a neuron input

Bias – Bi – a constant offset to all Neuron inputs. This determines the neuron output threshold.

Sensitivity – Si – scales the weight adjustment by effect on final output

L – number of layers where Li is the current layer

# Source Code

## Program Flow

Execution starts in dunebuggy.py. There is no main so execution starts at the top.

A map is drawn. The map file is always called SimMap so that other maps may be substituted.

Events required for DuneBuggy control are enabled. The main loop looks for events:

* QUIT – exit the program
* MOUSEBUTTONDOWN – check\_menu is called to find out if the mouse is down on a menu button. If so, the text in the button is returned. Then a sim\_map call is made to respond to the action. If the cursor is over the map then a hill is drawn or replaced with grass.
* KEYUP – the up arrow key is pressed. The vehicle is moved up.
* KEYDOWN – the down arrow key is pressed.
* nextMove is called so that the AI can move the vehicle
* The display is updated. It may show changes
* The frames\_per\_second clock puts the loop to sleep for a short time (see parameter fps)

## Files

The following files are the Python source code:

* Dunebuggy.py – main loop where execution starts and the user interface is monitored
* Desertmap.py – the map and menu are drawn for the current simulator screen
* Button.py – definition and management of the user interface menu buttons
* Vehicle.py – management of the AILab vehicle
* Sensors.py – process the location of hills and the exit as input to the AI
* nextMove.py – the adaptive AI that computes the vehicle path

The following files are for support:

* Grass.png – green to represent grass (16x16 pixels)
* Hill.png – picture of a hill (16x16 pixels)
* DBn.png – picture of the vehicle (16x16 pixels)

## File DuneBuggy.py

### Program Start

Execution starts in module dunebuggy.py. There is no main() so execution starts at the top of this code.

### Description

This creates the screen, sets up the map in the screen, adds the menus and then goes into a main loop. The main loop check for user events then moves the simulation forward by one step. The step rate is controlled by a timer.

## File Desertmap.py

### Description

Creates and manages the screen. This is where the menu buttons are added. Desertmap is the place where map can be changed.

### Classes

class SimMap - draws the frame and menu

class Help - display help file in multiple screens

class Menu - display the menu, return user selections

## File Button.py

### Description

Creates and supports a clickable button object for a menu

### Class Button

Create and support actions for a rectangular button. The border is shown when the button is selected.

#### Method Button.Init

Parameters:

* text – the text displayed in the button
* screen – the drawing area of DuneBuggy on the display. Top left corner is pixel 0,0.
* x – left pixel column of the button in the screen
* y – top pixel row of the button in the screen
* w – width of the button in pixels
* h – height of the button in pixels
* border\_size – width of the border around the outer edge of the button when the button is pressed.
* base\_color – color of the button

The button is drawn to the initial set of parameters.

#### Method Button.draw

Desc: blit the button on top of the border, color is current\_color. Note a possible extension here is to change the color when the button is pressed.

Parm: border\_width – number of pixels in the border if pressed, 0 if not pressed.

#### Method Button.check\_down

Desc: MOUSEBUTTONDOWN – checks if the mouse is within the button area. If it is the border is drawn.

Sets - border to wider (text block size reduced) to border\_width.

Returns - button text if pressed, else None. The text is used in DuneBuggy.py to execute the associated function.

#### Method Button.check\_up

Desc: MOUSEBUTTONUP – If the mouse is within the button area this restores border the button picture.

Sets - return to border width of 0

Returns - None

## File Vehicle.py

### Description

Define and manage a vehicle sprite.

The user or AI will move the vehicle across the map. The vehicle is a 20x20 pixel field (a sprite) drawn in Windows Paint. The location of the vehicle center is a complex vector. A change in location is a floating-point pair of x and y pixels offset from the 0,0 position at the upper left of the map. When the vehicle position is displayed the x and y values are rounded off to the nearest pixel on the map. Therefore, a small update to the location may not move the vehicle on the display. However, changes are additive so if enough direction change is added to the x axis then you will see the vehicle move to the right.

A complex vector supplied as input to the vehicle causes it to move in the direction of the vector. The value of the real part of the complex vector is the number of pixels the vehicle will move along the horizontal x axis. The real part is the x axis, the imaginary part is the y axis.

### Method: Vehicle init

Parameters:

* screen – the screen object that displays the map

The vehicle is a sprite so first inherit a sprite parent.

### Class Vehicle

The vehicle is a sprite. This is displayed on the screen via a picture (.pgn) file.

It's a Sprite so inherit the Sprite class. Returns: vehicle object. This is a subclass of class pygame.sprite.Sprite. This gives it the support to display and move a sprite on the screen.

#### method: vehicle.init

Parameters:

* screen – pygame.display.get\_surface – the drawing surface

#### method: vehicle.move\_by\_one

Responds to keyboard arrows to move the vehicle by 1 pixel in one of the four directions. The direction is converted to a complex value that is passed to method move.

Parmeters:

* direction – text of right, left, up, or down

#### method: move

Adds a complex value to the current location.

Parameters:

* amount – complex value indicating the direction and amount of location change

Replaces the previous vehicle display with background then show the vehicle at the new location.

## Class Hills

Define a hill as an inner class object. Must move/remove them at an x,y. This class tracks all hills in a list by x,y location and state. If state is 'inUse' then it is being displayed. If state is 'free' it exists but is not shown.

#### method: init

Parameters:

* screen– pygame.display.get\_surface – the drawing surface

#### method: Hills.add

Create a hill at location x,y. Hill is a .pgn picture of size 20x20. Uses the hill in the hills list that has a state of free. Else appends the new hill to the hill list.

Parameters:

* location – the hill location, a complex number

#### method:H ills.remove

Covers a hill on the screen with background. Sets state in the hills list to 'free'.

## Class Hill

Creates a new hill and sets it initial state to 'inUse'.

## Def load\_png

Loads an image and converts it to a rectangle of pixels.

Parameters:

* name – the name of the file in the current directory. Include the file type of .png.

Returns:

A tuple of image and rectangle of the image.

## File DriverAI.py

### Description

This is the AI that determines the direction and distance for the next move of the vehicle.

### Class DriverAi

#### Method: compute

Calculate the next direction.

#### Method: forward

Forward propagation through our network

Returns the output of the network limited to +/- 1, 1j

#### Method: backward

Backward propgate through the network

#### Method: sigmoid

Activation function applies the sigmoid factor to each weight limiting the range of the weight to +-1,+-1j.

#### Method: sigmoidPrime

Derivative of sigmoid.

#### Method: getWeights

Returns the current set of weights for backup.

#### Method: setWeights

Sets the weights to a backup set of weights.

## File Sensors.py

### Description

This sets the sensors on the vehicle. This code sets the number of sensors on the vehicle and the angle each sensor points.

A sensor acts like an antenna with a reflector so it only senses forward, not behind the antenna. The 3db angle (half-power angle) of each sensor is set in the definition of the sensor. Initially it is set to 45 degrees. The field of view falls off to 0 at +/- 90 degrees. This determines sensitivity as a function of angle and amount of overlap with the other sensors. As an example set sensors with one pointing straight ahead (0 degrees) and the other two at 45 degrees to each side.

The sensors detect hills by reflection of a transmitted signal. The power received for a signal is increased by the square of the distance – it increases by 3db for each halving of the distance. The sensor can report the angle and distance of each hill. The exit transmits an encrypted signal which enables a sensor to filter out the exit signal from the combination of all signals. Note that a sensor reports the sum of all signals it can sense.

### Classes

## File Tools.py

### Description

### Class Debug

#### Method init

#### Method

### Class Write

#### Method init

#### Method

### Class Help

#### Method init

#### Method display

#### Method draw

#### Method get\_surfaces

### Class UnitTest

#### method

# Development Process

## Language

### Language is Python 3.7.0.

## Library pygame

This public library provides the tools for displaying and managing objects on a screen.

### Install pygame.

<https://www.pygame.org/wiki/GettingStarted#Windows%20installation>

Open a DOS window and enter:

py -m pip install -U pygame –user

Then to check it out:

py -m pygame.examples.aliens

There are over 500 games at that site.

## Library numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. [Wikipedia](https://en.wikipedia.org/wiki/NumPy)

It is provided at [www.scipy.org/scipylib/download.html](http://www.scipy.org/scipylib/download.html).

In dunebuggy it is used to enable the AI to use complex vectors describing distance and angle to objects and also performing AI operations with complex weights.

## Editing

The files were edited with the Notepad++ text editor.

## Execution

To run the python code start the python shell. Load the main file: dunebuggy.py. Select Run/Run Module or just use key F5.

## Backup

Program code is backed up on github and project dunebuggy. First backup on 17 April 2018.